1. **Explain about Spring Security and its history?**

   Spring Security is a framework which provides various security features like: authentication, authorization to create secure Java Enterprise Applications.

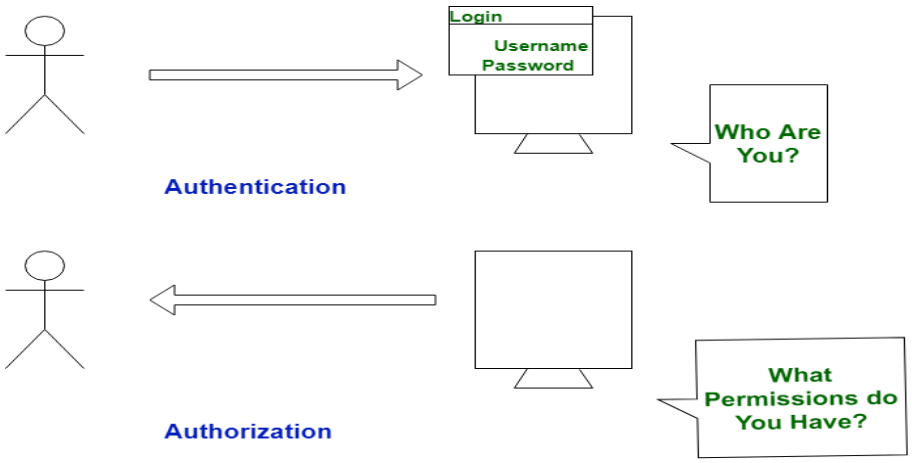
   It is a sub-project of spring framework which was started in 2003 by Ben Alex. Later on, in 2004, It was released under the Apache License as Spring Security 2.0.0.

   It overcomes all the problems that come during creating non spring security applications and manage new server environment for the application.

2. **Difference between Authentication and Authorization?**

   Authentication: is about validating your credentials like User Name/User ID and password to verify your identity.

   Authorization: is the process to determine whether the authenticated user has access to the particular resources.



3. **What are all advantages of Spring Security?**

   - ✓ Spring Security is an open source security framework
   - ✓ Comprehensive support for authentication and authorization.
   - ✓ Integration with Spring MVC
   - ✓ CSRF protection
   - ✓ Java Configuration Support.

4. **What is the latest version of spring Security and its features?**

   Latest version is 5.4.5

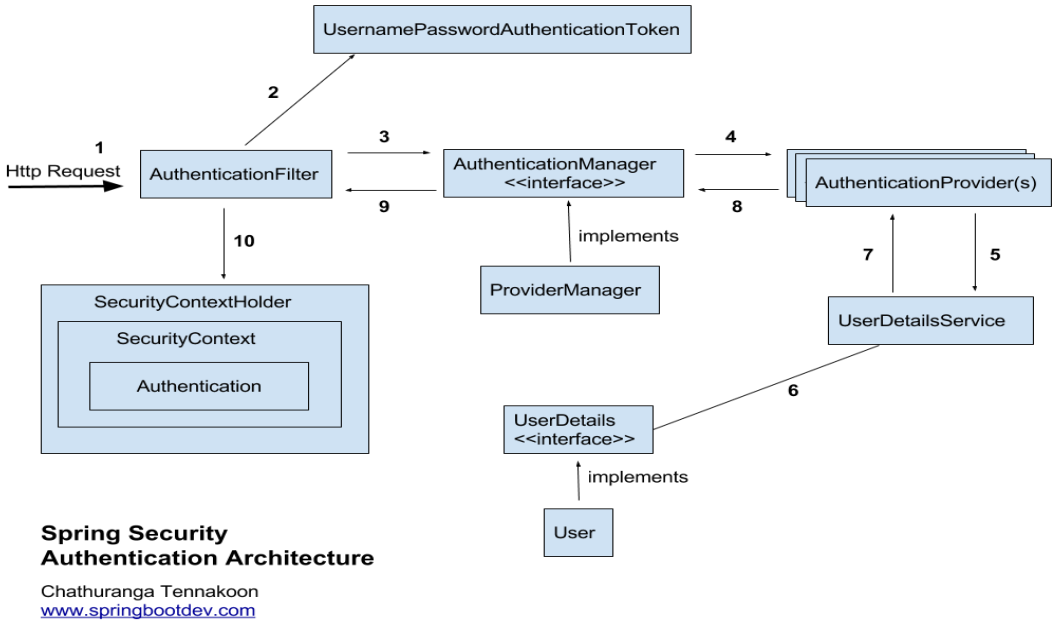
   - ✓ OAUTH 2 updates

- ✓ Logging update
- ✓ LDAP update (Added support for configuring application with a random port)

5. Explain the features of Spring Security?

- ✓ **SSO:** This feature allows a user to access multiple applications with the help of single account (user name and password).
- ✓ **LDAP (Lightweight Directory Access Protocol):** It is an open application protocol for maintaining and accessing distributed directory information services over an Internet Protocol.
- ✓ **Basic Access Authentication:** Spring Security supports Basic Access Authentication that is used to provide user name and password while making request over the network.
- ✓ **Digest Access Authentication:** This feature allows us to make authentication process more secure than Basic Access Authentication. It asks to the browser to confirm the identity of the user before sending sensitive data over the network.
- ✓ **Web Form Authentication:** In this process, web form collect and authenticate user credentials from the web browser. Spring Security supports it while we want to implement web form authentication.
- ✓ **Authorization:** Spring Security provides this feature to authorize the user before accessing resources. It allows developers to define access policies against the resources
- ✓ **Remember-me:** Spring Security supports this feature with the help of HTTP Cookies. It remember to the user and avoid login again from the same machine until the user logout.

6. Explain Spring Security Architecture?



AuthenticationFilter: This is the filter that intercepts requests and attempts to authenticate it. In Spring Security, it converts the request to an Authentication Object and delegates the authentication to the AuthenticationManager.

AuthenticationManager: It is the main strategy interface for authentication. It uses the lone method authenticate() to authenticate the request. The authenticate() method performs the authentication and returns an Authentication Object on successful authentication or throw an AuthenticationException in case of authentication failure. If the method can't decide, it will return null. The process of authentication in this process is delegated to the AuthenticationProvider which we will discuss next.

AuthenticationProvider: Here is this class can use the loadUserByUsername() method of the UserDetailsService implementation. If the user is not found, it can throw a UsernameNotFoundException.

UserDetailsService: It is most commonly used in database backed authentication to retrieve user data. The data is retrieved with the implementation of the lone loadUserByUsername() method where we can provide our logic to fetch the user details for a user. The method will throw a UsernameNotFoundException if the user is not found.

PasswordEncoder: Until Spring Security 4, the use of PasswordEncoder was optional. The user could store plain text passwords using in-memory authentication. But Spring Security 5 has mandated the use of PasswordEncoder to store passwords. This encodes the user's password using one its many implementations. The most common of its implementations is the BCryptPasswordEncoder.

Spring Security Context: This is where the details of the currently authenticated user are stored on successful authentication. The authentication object is then available throughout the application for the session. So, if we need the username or any other user details, we need to get the SecurityContext first. This is done with the SecurityContextHolder, a helper class, which provides access to the security context. We can use the setAuthentication() and getAuthentication() methods for storing and retrieving the user details respectively.

7. What is the use of PasswordEncoder?

The PasswordEncoder is an interface in Spring security that provides password encoding or password hashing. It has two methods encode() to encode the raw password and matches() to verify the encoded password obtained from the database matches the submitted raw password after it too is encoded using the same salt and same hashing algorithm.

8. What are some implementations of PasswordEncoder in Spring Security?

The two important implementations of the new PasswordEncoder interface are BCryptPasswordEncoder and the confusingly named StandardPasswordEncoder based on SHA-

256. The BCrypt implementation is the recommended one. There is also a NoOpPasswordEncoder which does no encoding. It's intended for unit testing only.

9. **What is salting in spring security?**

   Salting secure your application from Dictionary-Attack. Using Salt you may add an extra string in password so hacker find it difficult for braking the password.

   There are 2 salt methods,

   Global Salt.
   Per User Salt.

   In Global Salt there is one single common word append to password. In Per User Salt we have to give one user attribute serve as Salt String.

10. **Is there a way to set up basic authentication and form login in same application?**

    Yes. We may need form login for web app and basic for rest services. In that case multiple http configuration is required.

11. **What is DelegatingFilterProxy in Spring Security?**

    DelegatingFilterProxy is the entry point of Spring Security in a Java web application.

    As already discussed, Spring Security is based on the concept of Servlet filters. So, DelegationFilterProxy is a Servlet Filter implementation which works as a root Filter.

```
<filter>
<filter-name>rootSecurityFilter</filter-name>
<filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
</filter>
<filter-mapping>
<filter-name>rootSecurityFilter</filter-name>
<url-pattern>/*</url-pattern>
</filter-mapping>
```

    By doing this, it also works as bridge between web application and Spring IoC container (ApplicationContext) and its lifecycle.

12. **Explain the structure of Authentication object in Spring Security?**

    Authentication is an interface.

    We have different types of objects depending upon different Authentication implementations.

All these implementations have 3 parts:

Principal
Credentials
Authorities

13. Explain spring security Example?

Add the Spring Security starter in pom.xml,

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-test</artifactId>
    <scope>test</scope>
</dependency>
```

Add the Security configuration file by add the below annotations,

- ✓ @Configuration
- ✓ @EnableWebSecurity

Spring security class should extends the WebSecurityConfigurerAdapter

```
@Configuration
@EnableWebSecurity
public class SecurityConfigurer extends WebSecurityConfigurerAdapter {
```

WebSecurityConfigurerAdapter will provide below methods

Authentication (Actual UserDetailsService call will happen from here and password to be encrypted here**)**:

```
@Override
protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    auth.userDetailsService(userDataService).passwordEncoder(passwordEncoder());
}

@Bean
public PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
}
```

Authorization (We can add the service URL's to be ignored after authentication and JWT authorization**):**

```
//Authorization
@Override
protected void configure(HttpSecurity http) throws Exception {
    http.csrf().disable().authorizeRequests().antMatchers("/jwt/token/v1/generateToken").permitAll().anyRequest()
            .authenticated().and().exceptionHandling().and().sessionManagement()
            .sessionCreationPolicy(SessionCreationPolicy.STATELESS);
    http.addFilterBefore(jwtFilter, UsernamePasswordAuthenticationFilter.class);
}
```

Static resources access:

```
//Giving the access to static resources like CSS/JS
@Override
public void configure(WebSecurity web) throws Exception {
      web
       .ignoring()
          .antMatchers("/resources/**");
}
```

From authentication method we will call the UserDetailsService class loadByUserName method to match the current user existence in database and it will return his details like Role

```
@Service
public class UserDataService implements UserDetailsService {

    @Autowired
    private UserRepository userRepository;

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

        UserEntity userEntity = userRepository.findByUserName(username);

        return new org.springframework.security.core.userdetails.User(userEntity.getUserName(),
                userEntity.getPassword(), new ArrayList<>());
    }
```

14. Explain Spring Security basic authentication example?

After adding the spring security dependency when we run the application we will see the one password generated by spring,

```
Using generated security password: 6c89f2be-5a14-46f5-b659-576e4ff5e47d
```

When we hit the local host url will see the below login screen

We need to enter the userId Id as **user** and password as **generated password**.

Instead of using the above default one we can define the user id and password in properties file,

interview-practice/pom.xml application.properties

```
server.port=8085

spring.security.user.password=test
spring.security.user.name=test
```

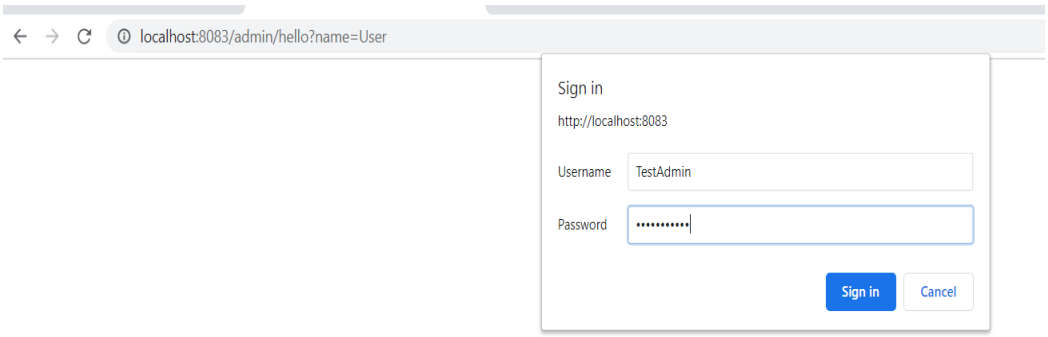
15. Explain about in memory authentication?

In memory authentication can happened by hard coding the user details in authentication method.

```
@Configuration
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.inMemoryAuthentication()
                .withUser("sergey")
                .password("{noop}12345678")
                .roles("USER")
                .and()
                .withUser("John")
                .password("{noop}87654321")
                .roles("MANAGER");
    }
```

16. Explain about spring digest authentication example?



17. Explain about method level Spring security?

Apart from authentication, spring security also check authorization of the logged in user. After login which user is authorize to access the resource is done on the bases of user's ROLE.

At the time of creating user in WebSecurityConfig class, we can specify user?s ROLE as well.

Security applied on a method restricts to unauthorized user and allow only authentic user.

```
// Only, a person having ADMIN role can access this method.
@RequestMapping(value="/update", method=RequestMethod.GET)
@ResponseBody
@PreAuthorize("hasRole('ROLE_ADMIN')")
public String update() {
   return "record updated ";
}
```

18. How handle the cors issue in Spring Security?

To enable CORS support through Spring security, configure CorsConfigurationSource bean and use HttpSecurity.cors() configuration.

```java
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.cors().and()
            //other config
    }

    @Bean
    CorsConfigurationSource corsConfigurationSource()
    {
        CorsConfiguration configuration = new CorsConfiguration();
        configuration.setAllowedOrigins(Arrays.asList("https://example.com"));
        configuration.setAllowedMethods(Arrays.asList("GET","POST"));
        UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
        source.registerCorsConfiguration("/**", configuration);
        return source;
    }
}
```

19. Explain about JWT?

JSON Web Token (JWT) is an open standard that defines a way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed. JWTs can be signed using a secret (with the HMAC algorithm) or a public/private key pair using RSA or ECDSA.

Will use it for authorizing the incoming requests.

JWT Structure: In its compact form, JSON Web Tokens consist of three parts separated by dots (.), which are

- ✓ Header
- ✓ Payload
- ✓ Signature

Therefore, a JWT typically looks like the following.

xxxxx.yyyyy.zzzzz

Header: The header typically consists of two parts: the type of the token, which is JWT, and the signing algorithm being used, such as HMAC SHA256 or RSA.

```json
{
  "alg": "HS256",
  "typ": "JWT"
}
```

Payload: This will contain the actual information.

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

Signature: To create the signature part, you have to take the encoded header, the encoded payload, a secret, the algorithm specified in the header, and sign that.

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret)
```

20. How to get the current logged-in username in Spring Security?

The object returned by getContext() is an instance of the SecurityContext interface. Normally, the getPrincipal() method returns the UserDetails object in Spring Security that contains all the details of the user that is currently logged in.

```
Object principal = SecurityContextHolder.getContext().getAuthentication().getPrincipal();

if (principal instanceof UserDetails) {
  String logged_in_username = ((UserDetails)principal).getUsername();
} else {
  String logged_in_username = principal.toString();
}
```

21. Why do we need SSL certificate for website?

The SSL certificate encrypts data that goes back and forth from the user's device to the target website. Any time a user enters information on your site, SSL ensures that they can navigate securely from their browser to your web server.

22. Difference between a Java Keystore JKS and PKCS12?

The default keystore format used was JKS until Java 8. However, now since Java 9, PKCS12 has been the default keystore format.
Another main difference between JKS and PKCS12 is that JKS is a Java-specific format, while PKCS12 stores encrypted private keys and certificates in a standardized and language-neutral way

23. How to secure the endpoints?

We can do it by using SSL certificates.

openssl genrsa -des3 –out priv.pem -passout pass:myPassword 1024